Compilation – TD2 : Activations – Exercice 1 : À la main

Nawfal ‘Massine’ MALKI – 4991 – STI 3A TD3

Question 1 : Dessiner l’état de la pile au point d’exécution //P

Programme 1 (step-by-step) :

- domaine de l’appelant du main()
- domaine du main()
- domaine de f()
- **état de la pile au point d’exécution //P**

1.void main()	2.int result;	3.result = f(1);	4.int f(int x)	5.int result;
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
		Arg : x = 1	Arg : x = 1	Arg : x = 1
			@retour	@retour
			Old bp du main	Old bp du main
				Slot 0 : result

6.[[x]]	7.[[1]]	8.result=x+1; //P
@retour	@retour	**@retour**
Old bp	Old bp	**Old bp**
Slot 0 : result	Slot 0 : result	**Slot 0 : result**
Arg : x = 1	Arg : x = 1	**Arg : x = 1**
@retour	@retour	**@retour**
Old bp	Old bp	**Old bp**
Slot 0 : result	Slot 0 : result	**Slot 0 : result = x+1**
x (=1)	x (=1)	
	1 (cst)	

Programme 2 (step-by-step) :

- domaine de l'appelant du main()
- domaine du main()
- domaine de pow()
- domaine de pow() (2ème appel)
- **état de la pile au point d'exécution //P**

1.void main()	2.int result;	3.[[pow(2,1)]]	4.int result;	5.[[n==0]]
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
		Arg : x = 2	Arg : x = 2	Arg : x = 2
		Arg : n = 1	Arg : n = 1	Arg : n = 1
		@retour	@retour	@retour
		Old bp	Old bp	Old bp
			Slot 0 : result	Slot 0 : result
				n = 1
				0 (cst)

6.[[n==0]]	7.jz else	8.[[x]]	9.[[x]]	10.[[n-1]]
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
Arg : x = 2	Arg : x = 2	Arg : x = 2	Arg : x = 2	Arg : x = 2
Arg : n = 1	Arg : n = 1	Arg : n = 1	Arg : n = 1	Arg : n = 1
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
0 (n==0)		x = 2	x = 2	x = 2
			Arg : x = 2	Arg : x = 2
				n = 1
				1 (cst)

11.[[n-1]]	12.pow(x,n-1)	13.int result;	14.[[n==0]]	15.[[n==0]]
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
Arg : x = 2	Arg : x = 2	Arg : x = 2	Arg : x = 2	Arg : x = 2
Arg : n = 1	Arg : n = 1	Arg : n = 1	Arg : n = 1	Arg : n = 1
@retour	@retour	@retour	@retour	@retour
Old bp	Old bp	Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result	Slot 0 : result
x = 2	x = 2	x = 2	x = 2	x = 2
Arg : x = 2	Arg : x = 2	x = 2	x = 2	x = 2
Arg : 0 (n-1)	Arg : 0 (n-1)	0 (n-1)	0 (n-1)	0 (n-1)
	@retour	@retour	@retour	@retour
	Old bp	Old bp	Old bp	Old bp
		Slot 0 : result	Slot 0 : result	Slot 0 : result
			n = 0	1 (n==0)
			0 (cst)	

16.jz invalide	17.[[1]]	18.result = 1; //P
@retour	@retour	@retour
Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result
Arg : x = 2	Arg : x = 2	Arg : x = 2
Arg : n = 1	Arg : n = 1	Arg : n = 1
@retour	@retour	@retour
Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result
x = 2	x = 2	x = 2
x = 2	x = 2	x = 2
0 (n-1)	0 (n-1)	0 (n-1)
@retour	@retour	@retour
Old bp	Old bp	Old bp
Slot 0 : result	Slot 0 : result	Slot 0 : result = 1
	1 (cst)	

Question 2 : Traduire la fonction pow en déroulant les règles de traduction vues en cours

```
[[
      int pow (int x, int n)
      {
            int result;
            if(n == 0)
                  result = 1;
            else
                  result = x * pow(x, n-1);
            return result;
      }
]]=

[[int result; if(n==0)result=1;else result=x*pow(x,n-1); return result;]]=

[[int result;]]
[[if (n==0) result=1; else result=x*pow(x,n-1); return result;]]=
      [[if (n==0) result=1; else result=x*pow(x,n-1);]]=
            [[n==0]]=
                  // position de l'arg n dans la pile p/r à l'old bp (3)
                  push ϱ(n) = push - 3
                  mpush 0
                  // test d'égalité
                  teste
            Jz else
            [[result=1;]]=
                  mpush 1
                  pop ϱ(result) = pop 0
            Jmp endif
            else:
            [[result=x*pow(n-1);]]=
                  [[x*pow(n-1);]]=
                        [[x]]
                        // position de l'arg n dans la pile p/r à l'old bp (3)
                        push ϱ(x) = push -4
                        [[pow(x,n-1)]]=
                        [[x]]=
                              push ϱ(x) = push -4
                        [[n-1]]=
                              [[n]]=
                                    push ϱ(n) = push -3
                              mpush 1
                              sub
                        call pow
                  mul
            pop ϱ(result) = pop 0
            endif:
      [[return result;]]=
            [[result]] = push 0
            //nombre d'arguments de pow
            iret 2
      free
ret 2
```